International Journal of Computational Intelligence in Control
Vol. 13 no. 3 (December, 2021)

Received: 05th April 2021 *Revised: 28th June 2021* *Accepted: 26th July 2021*

Short Term Load Forecasting using A Novel Recurrent Neural Network

Sanjib Mishra & Sarat Kumar Patra
NIT Rourkela, *E-mails: sanjib.mishra77@gmail.com E-mail: skpatra@nitrkl.ac.in*

Abstract: *Short term load forecasting is essential to the operation of electricity companies. It enhances the energy-efficient and reliable operation of power system. Neural networks (NNs) have powerful nonlinear mapping capabilities. Therefore, they have been used to deal with predicting, in which the conventional methods fail to give satisfactory results. A novel Recurrent neural network (RNN) is proposed in this paper. Many types of computational intelligent methods are available for time series prediction. The novelty of this RNN lies in the usage of neurons instead of simple feedback loops for temporal relations. There is flexibility to use any type of activation functions in both feed forward and feedback loops. Number of hidden neurons can be changed on case to case basis for maximum accuracy. The performance of the RNN is demonstrated to be better than several other computational intelligent methods available.*

Index Terms: *Short term load forecasting, recurrent neural network, computational intelligence*

I. INTRODUCTION

Short term load forecasting is a time series prediction problem. It analyzes the pattern of future electrical load. The information is crucial to determine hydro-thermal generation mixture, to allot transmission corridor, to decrease over all loss of grid, and to increase operational efficiency.

The load is decomposed into two components. One is weather dependent, and the other is weather independent. Each component is modeled separately and the sum of these two gives the total load forecast. The behavior of these two controls the total load pattern. The behavior of weather independent load is mostly represented by Fourier series or trend profiles in terms of the time functions. The weather sensitive portion of the load is arbitrarily extracted and modeled by a predetermined functional relationship with weather variables.

Time series nonlinear predictors can be formed by placing zero-memory nonlinearity within the output stage of classical linear predictor. The nonlinearity is restricted to the output stage, as in a single layer neural network realization. On the other hand, if the nonlinearity is distributed through many layers of weighted interconnections, the concept of neural networks is fully exploited and more powerful nonlinear predictors may ensue. For the purpose of prediction, memory stages may be introduced at the input or within the network. In the prediction of hourly load, the network will have only one output neuron with a predicted value. For a dynamic system, such as a recurrent neural network for prediction, the state represents a set of quantities that summarizes all the information about the past behavior of the system that is needed to uniquely describe its future behavior.

The provision of feedback with delay introduces memory to the network and so is appropriate for prediction in case of recurrent neural networks. The feedback within it can be achieved either a local or global manner. The local feedback is achieved by the introduction of feedback within the hidden layer, whereas the global feedback is produced by the connection of the network output to the network input as shown in figure 1. Inter neuron connections are also possible. The use of the large number of tapped delay feedback input increases the input dimension, resulting in increased dimensionality problem.

Furthermore, the recurrent systems can inherently produce multistep ahead predictions; so, the multistep ahead prediction models, which are required in some process control applications, such as predictive control, can efficiently be built by RNNs [1]. Thus, the RNNs have attracted great interest. The Hopfield [2], the Elman [3], the Jordan [4], the fully recurrent [5], the locally-recurrent [6], the recurrent radial basis function [7], and the block-structured recurrent [8] networks are some of the examples of RNNs. In these structures, the feedback weights, assumed to be unity, are not trainable. The Hopfield network [2] is a simple recurrent network which has a fully connected single-layer structure. It is capable of restoring previously learned static patterns from their corrupted realizations. Elman [3] and Jordan [4] proposed specific recurrent networks which have an extra set of context nodes that copy the delayed states of the hidden or output nodes back to the hidden layer neurons. In these structures, the feedback weights, assumed to be unity, are not trainable. The fully recurrent neural network [5] allows any neuron to be connected to any other neuron in the network. While being more general, it lacks stability. In [6], the local feedback has been taken before the entry into the nonlinearity activation function. In [7], the past output values of a radial basis function network are fed back to both the network input and output nodes. In [8], a systematic way to build networks of high complexity using a block notation was given. The fully recurrent neural network allows any neuron to be connected to any other neuron in the network. While being more general, it lacks stability.

In this paper, the architecture and training procedure of a new RNN useful for short term load prediction / forecasting is presented. The structure of the proposed RNN differs from the other RNNs in the literature. The main difference of the proposed network compared to the available RNNs is that the temporal relations are provided by means of neurons arranged in three feedback layers, not by simple feedback elements, in order to enrich the representation capabilities of the recurrent networks. The feedback signals are processed in three feedback layers which contain neurons as in feedforward layers. In these feedback layers, the weighted sums of the delayed outputs of the hidden and output layers are passed through activation functions and applied to the feedforward neurons via some adjustable weights.

Following this introduction the remaining paper is organized as under. Section II provides details of proposed recurrent neural network while Section III analyzes the input & output parameters. The experimental results are presented in Section IV. Section V provides concluding remarks.

II. PROPOSED RECURRENT NEURAL NETWORK

The RNN architecture used here is presented in figure 2, where *Input*(k) & $y(k)$ represents the input and output of the RNN, respectively, and k is the time index. The RNN has three feedforward and feedback layers. In the feedforward layers, W_1 & W_2, represent the weights between the input and hidden layers, and the hidden and output layers, respectively. In addition to the feedforward layers, the RNN has two local and one global feedback layers. In these feedback layers, the weighted sums of the delayed outputs of the hidden and output layers are

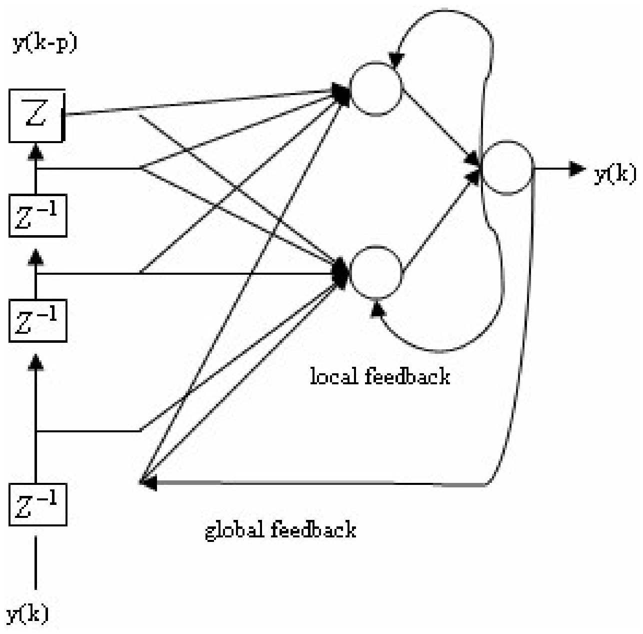

Figure 1: Structure of a Recurrent Neural Network with Local and Global Feedback

applied to certain activation functions as in the feedforward layer neurons. W_{b1}, W_{b2} & W_{b3} represent the weights connected to the inputs of the feedback layer neurons and z^{-1} represents the time delay operators. The outputs of the feedback layers neurons $h_c(k)$, $y_c(k)$ & $z_c(k)$ are applied to the hidden and output layers neurons via the adjustable weights W_{c1}, W_{c2}, W_{c3}.

The number of hidden neurons in this case is taken as two but should be tuned as per requirement of individual problem requirements. The number of neurons in the feedback layer from the hidden-to-hidden layer is set equal to the number of the hidden layer neurons i.e. two. The number of neurons in the feedback layer from the output-to-hidden layer is set equal to the number of the output layer neurons i.e. one. The number of neurons in the feedback layer from the output-to-output layer is set equal to the number of the output layer neurons i.e. one. However, their numbers can be adjusted to improve the accuracy on case to case basis. The numbers were finalized on trial and error.

Since the weights are updated by the back propagation method, the calculation of the Jacobian matrix is required. The backward phase computations from $k = T$ to $k = 1$ are performed by means of the back propagated path values of the MFLNN. When the forward and backward phases of the computations are completed, the sensitivities for each weight, which form the Jacobian matrix, are obtained as in the back propagation algorithm.

As it was expressed previously, the elements of the Jacobian matrix are computed in two stages which are referred to as the forward and backward phases. In the forward phase, the RNN actions are computed and stored from $k = 1$ to $k = T$ through the trajectory. The errors at every k are determined as the differences between the desired outputs and the RNN outputs. The initial values for the output of the hidden layer (h) and of the output layer (y) are set to 0. The net quantities produced at the input of the activation functions of the feedback neurons are

$$h(0) = 0, y(0) = 0$$
$$forout_{ch}(k) = [W_{b1} * h(k-1)] + B_{b1}$$
$$forout_{cy}(k) = [W_{b2} * y(k-1)] + B_{b2}$$
$$forout_{cz}(k) = [W_{b3} * y(k-1)] + B_{b3}$$

Where W_{b1}, W_{b2} & W_{b3} the input weights of the feedback layers and B_{b1}, B_{b2} & B_{b3} are the biases of the feedback layer neurons. The outputs of the feedback layer neurons h_c, y_c & z_c are computed by

$$h_c = \tanh\ (forout_{ch}(k))$$
$$y_c = \tanh(forout_{cy}(k))$$
$$z_c = \tanh(forout_{cz}(k))$$

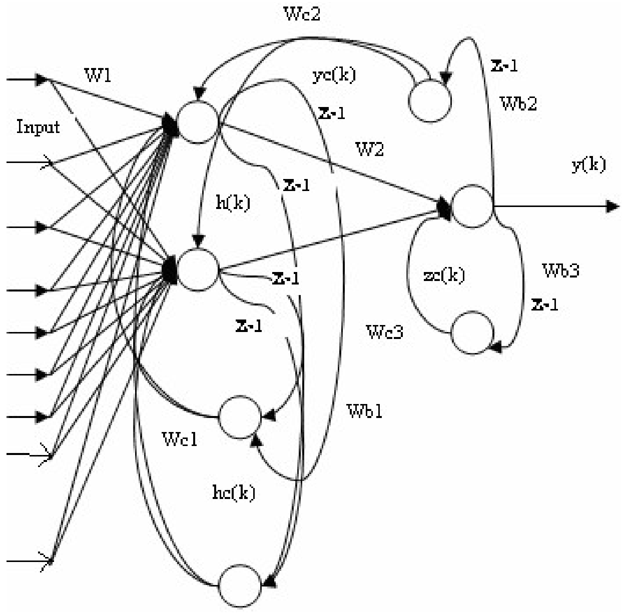

Figure 2: Structure of the Proposed RNN

Where tanh represent the activation functions of the feedback layer neurons. The net quantities *forout* of the hidden layer neurons and their outputs (h) are computed by

$$forout_h(k) = [W_1 * Input(k)]^T + \begin{bmatrix} [W_{c1} * h_c(k)] + \\ [W_{c2} * y_c(k)]^T + B_1 \end{bmatrix}$$
$$h(k) = \tanh(forout_h(k))$$

Where W_1 represents the weights between the input and hidden layers, and B_1 the biases applied to the hidden layer neurons. W_{c1} & W_{c2} are the weights of the feedback layers. tanh represents the hidden layer activation functions. Similarly, the net quantities $forout_y$ of the output layer neurons and their outputs (y)are computed by

$$forout_y(k) = [W_2^T * h(k)] + [W_{c3}^T * z_c(k)] + B_2$$
$$y(k) = purelin(forout_y(k))$$

Where W_2, B_2 & tanh represent the weights between the hidden and output layers, the biases applied to the output layer neurons, and the output layer activation functions, respectively. W_{c3} represents the output weights of the feedback layer. The error signal (e) is defined as the difference between the RNN output (y) and the desired output (*Output*).

$$e(k) = y(k) - Output(k)$$

The weights are adjusted to minimize the error (e), so the sensitivities with respect to each weight have to be computed. At every (k), the sensitivity for each weight is computed by multiplying the input of this weight in the RNN and the back propagated path, so the inputs of the weights in the back propagated path have to be computed. Therefore, after completing the forward phase computations, the backward phase computation is carried out through the back propagated path of RNN from $k = T$ to $k = 1$. The local sensitivities at $k = T + 1$ are set to 0.

$$\delta_{c3}(T+1) = 0$$
$$\delta_{c2}(T+1) = 0$$
$$\delta_{c1}(T+1) = 0$$

The local sensitivities are obtained as

$$\delta_2(k) = \sec h^2(forout_y(k)) * \begin{bmatrix} 1 + (W_{b2} * \delta_{c2}(k+1)) + \\ (W_{b3} * \delta_{c3}(k+1)) \end{bmatrix}$$
$$\delta_1(k) = \sec h^2(forout_h(k)) * \begin{bmatrix} (W_{b1} * \delta_{c1}(k+1)) + \\ (W_2 * \delta_2(k)) \end{bmatrix}$$
$$\delta_{c3}(k) = \sec h^2(forout_{cz}(k)) * [W_{c3} * \delta_2(k)]$$
$$\delta_{c2}(k) = \sec h^2(forout_{cy}(k)) * [W_{c2} * \delta_1(k)]$$
$$\delta_{c1}(k) = \sec h^2(forout_{ch}(k)) * [W_{c1} * \delta_1(k)]$$

Then, the sensitivity for each weight is computed by multiplying the values scaled by this weight in the RNN and the back propagated path as follows:

$$\frac{\partial e(k)}{\partial W_2} = \delta_2(k) * h\ (k)$$

$$\frac{\partial e(k)}{\partial B_2} = \delta_2(k)$$

$$\frac{\partial e(k)}{\partial W_1} = \delta_1(k) * Input\ (k)$$

$$\frac{\partial e(k)}{\partial B_1} = \delta_1(k)$$

$$\frac{\partial e(k)}{\partial W_{c3}} = \delta_2(k) * z_c^{\ T}(k)$$

$$\frac{\partial e(k)}{\partial W_{b3}} = \delta_{c3}(k) * y^T(k-1)$$

$$\frac{\partial e(k)}{\partial W_{c2}} = \delta_1(k) * y_c^{\ T}(k)$$

$$\frac{\partial e(k)}{\partial W_{b2}} = \delta_{c2}(k) * y^T(k-1)$$

$$\frac{\partial e(k)}{\partial W_{c1}} = \delta_1(k) * h_c^{\ T}(k)$$

$$\frac{\partial e(k)}{\partial W_{b1}} = \delta_{c1}(k) * h^T(k-1)$$

$$\frac{\partial e(k)}{\partial B_{b3}} = \delta_{c3}(k)$$

$$\frac{\partial e(k)}{\partial B_{b2}} = \delta_{c2}(k)$$

$$\frac{\partial e(k)}{\partial B_{b1}} = \delta_{c1}(k)$$

Then network weights & biases can be calculated as follows:

$$W_{b1} = W_{b1} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{b1}}\right]$$

$$B_{b1} = B_{b1} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial B_{b1}}\right]$$

$$W_{c1} = W_{c1} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{c1}}\right]$$

$$W_{b2} = W_{b2} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{b2}}\right]$$

$$B_{b2} = B_{b2} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial B_{b2}}\right]$$

$$W_{c2} = W_{c2} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{c2}}\right]$$

$$W_{b3} = W_{b3} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{b3}}\right]$$

$$B_{b3} = B_{b3} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial B_{b3}}\right]$$

$$W_{c3} = W_{c3} - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_{c3}}\right]$$

$$W_1 = W_1 - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_1}\right]$$

$$B_1 = B_1 - \left[\mu * e(c) * \frac{\partial e(k)}{\partial B_1}\right]$$

$$W_2 = W_2 - \left[\mu * e(c) * \frac{\partial e(k)}{\partial W_2}\right]$$

$$B_2 = B_2 - \left[\mu * e(c) * \frac{\partial e(k)}{\partial B_2}\right]$$

(μ) is the learning rate of the RNN. “Purelin” & “logsig” activation functions are used in this simulation.

III. INPUT & OUTPUT FOR THE RNN MODEL

In our analysis, the ANN model uses nine inputs, which constitute the load at hour ‘hr-1’ , ‘hr-2’, ‘hr-3’ of same day, ‘hr’, ‘hr-1’, ‘hr-2’ of previous day, & ‘hr’, ‘hr-1’, ‘hr-2’ of same day of previous week. Only one output node is used representing a 24-hour ahead load forecast at hour ‘hr’ in the lead time.

The reason behind taking the specific inputs are as follows: It takes into consideration the hour of the day effect to map hourly load variation. Day of the week is taken into account to map weekly pattern of industrial and commercial load pattern on week days and weekends. Seasonal variation is gradual so previous day load pattern as an explicit input takes care of seasonal mapping.

IV. SIMULATION RESULTS

The acceptable criteria for a particular model is based upon the (i) mean average percentage error (MAPE), (ii) number of hours in which it gives negative MAPE, (iii) time taken by the model to get trained. The acceptable criteria (i) & (iii) are self explanatory. The second criteria signifies the under estimation of required load. Under estimation of load may stress the generation units. The performance of the proposed system has been compared with performance of other soft computing techniques using same training data set.

The minimum mean average percentage error (i.e. MAPE) in case of Back propagation trained Multi Layer Perceptron Neural Network (BP-MLP), was found to be 2.5512 % with logsig activation function with a network using 17 hidden neurons, learning rate of 0.1, & Guyen-Widrow parameter initialization [9].

In case of Genetic Algorithm trained Multi Layer Perceptron Neural Network (GA-MLP), the best result was found to be, MAPE of 2.1331 %, with 4 hidden neurons, and tansig activation function.

In case of Particle Swarm Optimization trained Multi Layer Perceptron Neural Network (PSO-MLP), the best MAPE of 2.0748 % was achieved using 4 hidden neurons using logsig activation function.

In case of Artificial Immune System trained Multi Layer Perceptron Neural Network (AIS-MLP), the best MAPE of 3.9869 % was achieved using 4 hidden neurons using logsig activation function.

In case of the proposed RNN, the best result was MAPE of 1.8515 %, with 2 hidden neurons and tanh activation function.

We have compared result with Adaptive Neuro Fuzzy Inference System (ANFIS), which provided best performance MAPE of 1.7505%.

Prediction performance of algorithms cited in Table.1 for different days of a year with ten days interval is tabulated in Table 2. The minimum value of MAPE for a specific forecasted day is highlighted in the color same as the corresponding algorithm highlighting color. Fig.3 describes the results graphically.

In Table 3, the prediction performance of the proposed RNN is tabulated taking into consideration type of initialization method & number of hidden neurons. The first row gives the formula / standard Matlab

Table 1
MAPE Comparisons

Network	*Best MAPE in%*
BP-MLP	2.5512
GA-MLP	2.1331
PSO-MLP	2.0748
AIS-MLP	3.9869
ANFIS	1.7505
Proposed RNN	1.8515

Table 2
MAPE for Different Algorithms

	MAPE in %						
Forecasted Day	*RNN*	*PSO*	*GA*	*BP*	*ANFIS*	*AIS*	*Min. MAPE in %*
38	2.5058	3.9252	3.2463	3.3999	2.6005	7.3829	2.5058
48	2.515	4.8058	4.1073	2.9817	4.8723	6.7753	2.515
58	2.8246	3.0832	4.1287	3.6983	4.1178	5.4296	2.8246
68	5.9872	5.7972	6.7795	10.857	7.1008	16.2978	5.7972
78	4.1836	6.584	6.1948	5.0064	9.4502	9.9884	4.1836
88	4.1054	3.3303	6.4879	4.8648	3.5845	11.089	3.3303
98	3.0897	6.0078	6.8629	4.6654	3.0966	15.0009	3.0897
108	2.2356	7.2016	6.596	4.4614	3.1853	11.3845	2.2356
118	5.1856	5.6647	7.3346	5.8641	2.8272	10.693	2.8272
128	5.135	8.382	7.2612	6.4135	5.6529	13.6429	5.135
138	4.7324	8.1828	6.8799	6.1848	5.1995	9.3278	4.7324
148	3.971	7.2797	6.3819	4.1229	3.4567	14.9945	3.4567
158	4.287	6.8941	7.9228	6.4008	4.8156	14.0065	4.287
168	4.8313	6.6235	5.4471	5.4926	3.351	9.9973	3.351
178	3.4942	7.7839	5.6693	4.8353	2.4007	8.3081	2.4007
188	5.5597	5.5501	6.534	6.0204	4.2112	10.2101	4.2112
198	2.9065	6.6366	5.2949	3.2805	2.9617	7.8238	2.9065
208	2.5477	2.3002	4.1691	3.6629	6.882	7.0529	2.3002
218	1.9966	2.8583	2.8516	2.8102	9.5208	3.9869	1.9966
228	2.5437	2.8844	3.4632	3.1102	2.7833	5.7865	2.5437
238	3.6476	3.4	5.0082	3.9777	2.0411	7.7786	2.0411
248	1.8015	2.4227	2.1331	2.6848	1.814	5.3147	1.8015
258	3.289	4.3223	3.6149	3.341	7.749	8.9318	3.289
268	2.5874	2.0748	3.1062	3.4481	1.7015	4.2912	1.7015
278	2.7438	2.5483	2.3901	2.5512	3.4771	6.6248	2.3901
288	2.0366	3.579	3.6706	2.8106	2.1296	8.0993	2.0366
298	3.2513	3.7053	3.5433	3.6867	6.3116	8.4539	3.2513
308	3.8914	3.175	4.9403	4.8761	3.9409	6.7971	3.175
318	2.9107	3.5821	4.9228	3.7574	2.9512	6.6712	2.9107
328	4.4412	5.1506	5.3382	4.944	4.491	6.721	4.4412
338	3.3859	3.0508	4.1621	3.3731	4.8864	6.7823	3.0508
348	3.4505	3.9801	4.3726	3.7919	2.2507	6.99	2.2507
358	3.8607	2.7123	4.5196	4.5042	3.2742	6.1525	2.7123

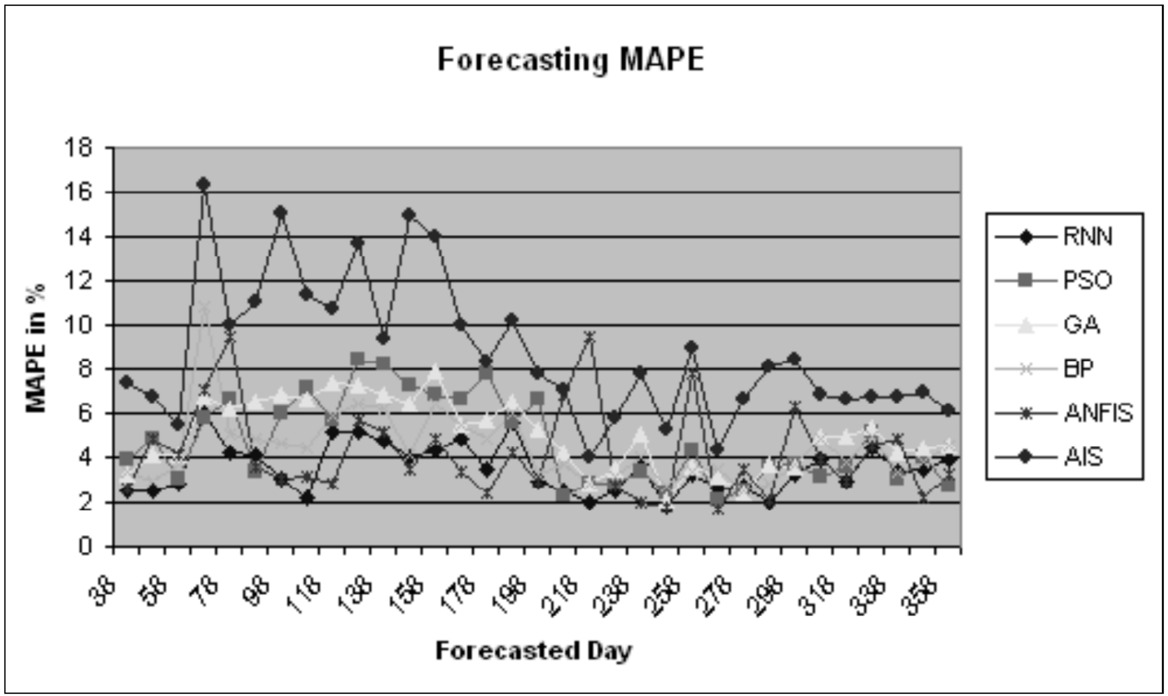

Figure 3: MAPE for Different Algorithms

Table 3
Performance Index Comparison of Proposed RNN Model

*A.*R and-(A/2) 9-2-1*	*A.*R and-(A/2) 9-3-1*	*2.*R and-1 9-2-1*	*2.*R and-1 9-3-1*	*Rands 9-2-1*	*Rands 9-3-1*	*Randnr 9-2-1*	*Randnr 9-3-1*	*Randnc 9-2-1*	*Randnc 9-3-1*	*Rand 9-2-1*	*Rand 9-3-1*	*NW 9-2-1*	*NW 9-3-1*	*Min. % Error*
4.171	4.107	3.126	4.416	3.126	4.41	-0.98	-4.29	4.03	4.53	4.57	5.15	3.92	3.80	0.98
4.416	4.243	4.546	3.668	4.546	3.66	-4.61	-15.1	2.80	7.05	7.31	6.53	5.80	4.60	2.80
3.842	3.515	6.368	4.002	6.368	4.00	-10.5	-30.7	4.68	9.889	11.99	10.54	8.846	5.51	3.51
0.989	0.614	5.646	2.384	5.646	2.38	-16.4	-41.6	4.84	10.77	13.90	16.14	9.311	4.50	0.61
-0.55	-0.75	4.754	1.220	4.754	1.22	-15.2	-36.1	5.62	8.545	11.46	14.82	8.899	3.93	0.55
-3.93	-4.02	-0.82	-2.98	-0.82	-2.9	-12.0	-23.6	0.05	2.311	2.363	4.191	1.475	-1.40	0.05
-1.85	-2.57	-0.69	-0.35	-0.69	-0.3	-12.9	-23.1	-0.59	-1.17	-1.76	-0.25	-2.31	-2.24	0.25
-2.00	-2.63	-3.72	-3.03	-3.72	-3.0	-18.5	-32.3	-3.79	-4.71	-5.62	-4.00	-6.60	-4.04	2.00
1.208	1.511	-2.50	-1.88	-2.50	-1.8	-20.2	-34.9	-1.92	-2.58	-3.21	-1.17	-4.45	-1.37	1.17
-3.44	-2.34	-6.85	-5.32	-6.85	-5.3	-29.6	-49.4	-6.69	-7.92	-8.94	-6.18	-10.3	-6.66	2.34
-0.17	0.630	-3.50	-4.45	-3.50	-4.4	-32.1	-60.6	-4.10	-5.70	-7.42	-3.78	-8.77	-4.88	0.17
0.270	1.182	-12.1	-4.55	-12.1	-4.5	-33.0	-53.2	-7.85	-7.41	-9.82	-12.7	-11.8	-5.44	0.27
5.886	6.720	0.189	2.279	0.189	2.27	-9.58	-20.4	1.77	2.923	3.052	0.439	1.300	2.75	0.18
2.471	3.608	3.308	1.162	3.308	1.16	-13.4	-32.6	1.53	6.802	8.479	5.692	5.554	2.36	1.16
2.920	3.243	7.879	4.327	7.879	4.32	-18.7	-47.3	7.09	13.19	17.58	18.73	12.34	7.24	2.92
2.757	2.736	8.930	5.127	8.930	5.12	-16.6	-42.9	8.66	13.11	18.43	22.72	14.08	8.44	2.73
-2.52	-2.72	4.374	-0.47	4.374	-0.4	-18.6	-42.1	5.85	9.596	10.63	14.39	8.851	2.67	0.47
-5.38	-5.88	-4.14	-5.59	-4.14	-5.5	-12.3	-21.6	-3.30	-1.19	-1.56	-0.35	-2.75	-3.64	0.35
-3.91	-5.93	-2.84	-3.05	-2.84	-3.0	-24.3	-43.6	-2.33	-3.41	-4.34	-1.67	-5.51	-3.44	1.67
-8.45	-8.47	-9.74	-6.34	-9.74	-6.3	-52.2	-90.4	-10.3	-12.5	-14.3	-9.05	-16.1	-11.2	6.34
1.047	1.290	-10.5	-6.26	-10.5	-6.2	-64.6	-110.	-8.89	-12.0	-15.0	-9.78	-18.5	-8.06	1.04

functions used to initialize the parameters (weight & bias) of neurons.

The value of A is taken as 0.72, Rand, Randnr, Rands, Randnc are standard Matlab random value generation functions. NW is Nguyen Widrow method of parameter initialization. The network is trained and tested with same set of historical data, so that we can select the parameter initialization method which will give the least Mean Average Percentage Error (MAPE). After the network is trained, it is subjected to testing data for prediction of next 21 (twenty one) hours load. The % prediction errors for each type of initialization method are delineated column wise under the respective initialization methods.

In Table 4, first row signifies summation of absolute percentage errors, second row gives number of minimum percentage errors provided by each method for a given testing data set, third row gives the number of negative % errors, fourth row gives the MAPE & fifth row gives the computation time required for testing for respective parameter initialization method.

Table 4
Percentage Error in Hourly Load Forecasting by Proposed RNN Model

*A.*R and-(A/2) 9-2-1*	*A.*R and-(A/2) 9-3-1*	*2.*R and-1 9-2-1*	*2.*R and-1 9-3-1*	*Rands 9-2-1*	*Rands 9-3-1*	*Randnr 9-2-1*	*Randnr 9-3-1*	*Randnc 9-2-1*	*Randnc 9-3-1*	*Rand 9-2-1*	*Rand 9-3-1*	*NW 9-2-1*	*NW 9-3-1*	
62.22	68.77	106.6	72.92	106.6	72.9	4.37E +02	8.57E +02	96.8	147.3	181.8	168.3	167.7	98.3	Total Abs. % Error
6	4	1	3	1	2	1	0	2	0	0	4	0	0	No. of Min. % Error
10	9	11	12	11	12	21	21	10	10	10	10	10	11	Negative % Error
12	11	5	9	5	8	1	0	7	5	3	5	4	6	Error less than 3%
2.963	3.275	5.079	3.472	5.07	3.47	20.81	40.82	4.61	7.01	8.65	8.01	7.98	4.68	Avg. hourly % Error
10.88	11.06	10.5	10.51	10.4	11.1	10.83	10.41	11.9	10.7	11.0	11.0	10.4	10.8	Compu-tation Time

V. CONCLUSION

The performance of the proposed RNN is compared with several other computational intelligence methods like multi layer perceptron neural network (MLPNN), MLPNN trained by GA, MLPNN trained by PSO, ANFIS to show the superiority in terms of accuracy of prediction. It has been shown that the proposed RNN achieves higher accuracy with less number of neurons.

The main advantages of the proposed RNN are as follows:

The temporal relations are provided by neurons, not be simple feedback paths, which enhance the nonlinear mapping capability.

It has a flexible feedback structure, so we can use different types of activation functions and different number of neurons on case to case basis for increasing accuracy.

REFERENCES

[1] D. P. Mandic, J. A. Chambers, Recurrent Neural Networks for Prediction. New York: Jhon Wiley & Sons, 2001.

[2] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci.*, **79**, 2554–2558, 1982.

[3] J. L. Elman, "Finding Structures in Time," *Cogn. Sci.*, **14**, 179–211, 1990.

[4] M. I. Jordan, "Supervised Learning and Systems with Excess Degrees of Freedom" COINS, Mass. Inst. Technol., Cambridge, MA, 1988, Tech. Rep. 88-27.

[5] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Comput.*, **1**, 270–280, 1989.

[6] A. C. Tsoi and A. D. Back, "Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures," *IEEE Trans. Neural Netw.*, **5**(2), 229–239, 1994.

[7] S. A. Billings and C. F. Fung, "Recurrent Radial Basis Function Networks for Adaptive Noise Cancellation," *Neural Netw.*, **8**(2), 273–290, 1995.

[8] S. Santini, A. D. Bimbo, and R. Jain, "Block-structured Recurrent Neural Networks," *Neural Netw.*, **8**(1), 135–147, 1995.

[9] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights," in *Proc. Int. Joint Conf. Neural Netw.*, **3**, pp. 21–26, 1990.